An Introduction to Constraint Programming

(For metaheuristic-minded researchers)

Andrea Roli

a.roli@unich.it

Dipartimento di Scienze

Università degli Studi "G. D'Annunzio" – Chieti-Pescara

Motivation

Why should I learn CP?

- One of the most efficient techniques for constraint satisfaction
- CP modelling useful also for metaheuristics
- Current research trend in applications: integration of CP and OR/AI techniques

Outline

- The path to CP...
  - Constraint Satisfaction Problems
  - Systematic search
  - Consistency techniques
- Constraint Programming
  - Modelling
  - Constraints
  - Search strategies

Constraint programming

- Roots in logic programming
- Nowadays, effective software technology for tackling combinatorial optimization problems
- Three main features:
  - **Modelling**: Based on the concept of *constraints*
  - **Propagation**: Efficient way to produce the consequences of a decision
  - **Search**: Efficient tree-based search algorithms

Constraints

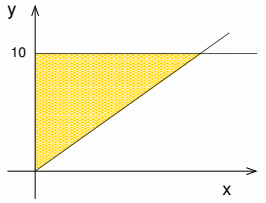
Logical relations among several variables

- **Declarative**
- **Additive**
- **Non directional**
- **Rarely independent**
- **Incremental operational behaviour**

Constraints

Additive: The order of imposition does not matter

E.g.: constraints $y \leq 10$ and $x \leq y$ ($x, y \in [0, 20]$, integer) define a set of feasible points, independently of the sequence we consider the constraints.



Constraints

Declarative: They do not specify a computational procedure to enforce the relationship

E.g.:the constraint *ordered*$([x_1, x_2, x_3, x_4])$ imposes the fact that variables x_1, x_2, x_3, x_4 should be numerically ordered.

The constraint does not specify an algorithm to enforce the property.

Constraints

Non directional: a constraint between X and Y can be used to infer information on Y given information on X and viceversa

E.g.: $x = y + 2$ specifies a *relation* between X and Y and it is NOT an assignment, i.e.,

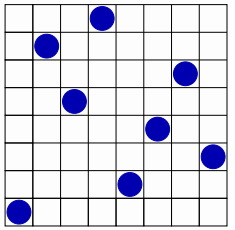
- If $y = 3$ the constraints forces x to assume value 5;
- If $x = 10$ the constraints forces y to assume value 8;

Constraints

- **Rarely independent**: They typically share some variables
- **Incremental operational behaviour**: Each time new information available, the computation does not start from scratch

Examples

N-Queens



Place N queens on a $N \times N$ chessboard in such a way that the queens cannot attack each other.

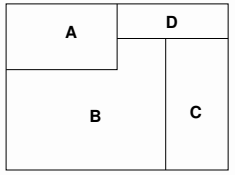
Constraint satisfaction

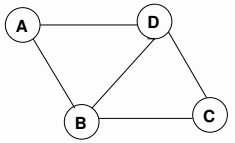
Constraint satisfaction problems (CSPs) deal with objects that has to be configured in such a way that all the constraints over them are satisfied.

- Puzzles
- Industrial configuration problems
- Assignment, scheduling, rostering, etc.
- Circuit testing and verification (e.g., SAT)
- . . .

Example

Map (graph) colouring





Find an assignments of colours to zones s.t. no two adjacent zones are coloured with the same colour.

Constraint satisfaction

A Constraint satisfaction problem (CSP) is defined by:

- variables $X = \{x_1, \ldots, x_n\}$
- variable domains $D_1, \ldots, D_n$ (finite)
- constraints among variables
- Goal: find a feasible assignment (or return infeasibility)
  Other goals: find all satisfiable assignments, count the number of feasible assignments

Modelling: examples

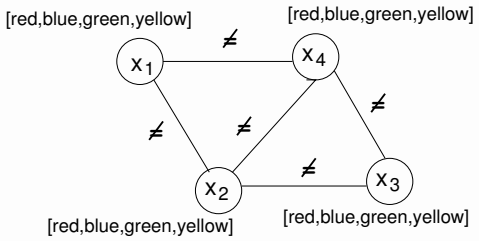
N-Queens

- Variables: $x_i \in \{1, 2, \ldots, N\}$ is the row of queen in column i
- Constraints:
  - $x_i \neq x_j, \forall i, j, i \neq j$
  - $x_i \neq x_j - (j - i), 1 \leq i < j \leq N$
  - $x_i \neq x_j + (j - i), 1 \leq i < j \leq N$

Note that in this model constraints are *binary*

Constraint graph

variables $\leftrightarrow$ nodes
constraints $\leftrightarrow$ (hyper)-arcs



Feasible solution: $x_1 = \texttt{red}, x_2 = \texttt{blue}, x_3 = \texttt{red}, x_4 = \texttt{green}$

Modelling: examples

N-Queens

- Variables: $x_i \in \{1, 2, \ldots, N\}$ is the row of queen in column i
- Constraints:
  - $x_i \neq x_j, \forall i, j$
  - $x_i - i \neq x_j - j, 1 \leq i < j \leq N$
  - $x_i + i \neq x_j + j, 1 \leq i < j \leq N$

Note that in this model constraints are *binary*

Modelling: examples

Map (graph) colouring

- Variables: $x_i \in \{1, 2, \ldots, N\}$ is the color of node i
- Constraints:
  - $x_i \neq x_j$ if `adjacent`(x_i, x_j)

Applications

- Scheduling
- Timetabling
- Resource Allocation
- Routing
- Packing - Cutting
- Graphics - Vision
- Planning

Constrained optimization

A Constrained optimization problem (COP) is defined by:

- variables $X = \{x_1, \ldots, x_n\}$
- variable domains $D_1, \ldots, D_n$
- constraints among variables
- the set of all possible feasible assignments $\mathcal{S}$
- *Objective function* $f : \mathcal{S} \rightarrow \mathbb{R}^+$
- Goal: find $s^* \in \mathcal{S}$ such that s^* optimizes the objective function.

Solving a CSP

- Systematic search (tree-based search)
- Consistency techniques

- CP combines both and improves upon them

Systematic search

Basic techniques:

- Backtracking based search
- Advanced tree-based search techniques

Chronological BT

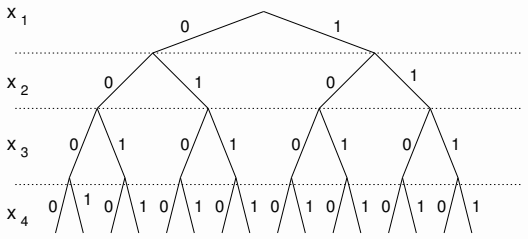
The idea:

1. Select an unassigned variable
2. Select a value within its domain
3. If consistent with constraints then goto 1
4. Else if still values not tried choose another value
5. Else backtrack to previous variable

Backtracking based search

- Incrementally attempts to extend a partial assignment by assigning a tentative value to a variable
- Tree based
- *A posteriori* use of constraints

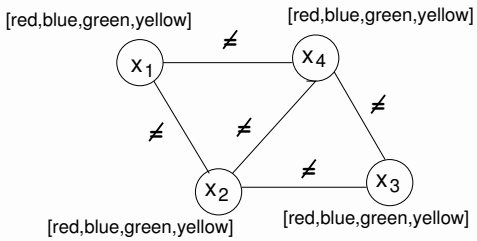
Chronological BT



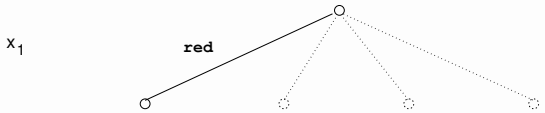
Example of a binary tree: chronological BT with static variable and value ordering explores it depth first

Example

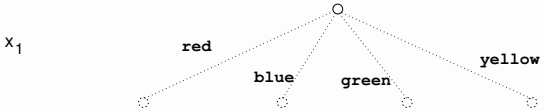
Graph colouring



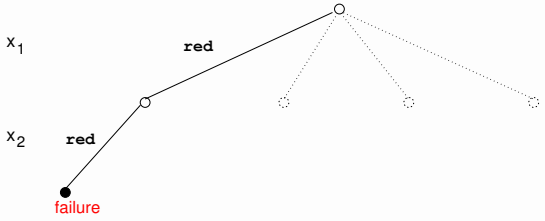
Example



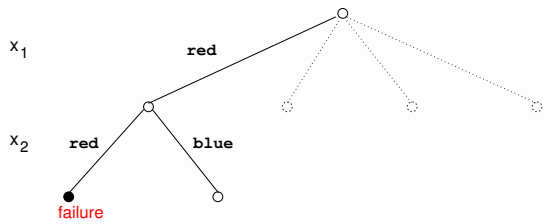
Example



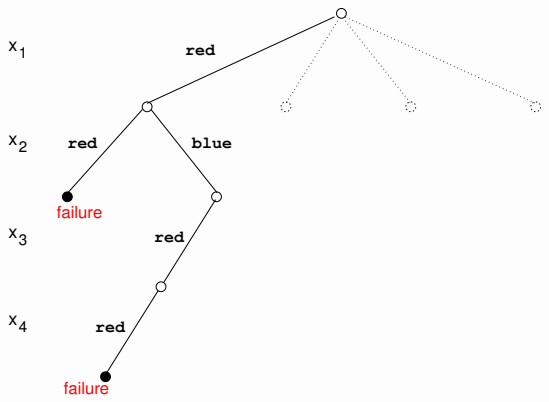
Example



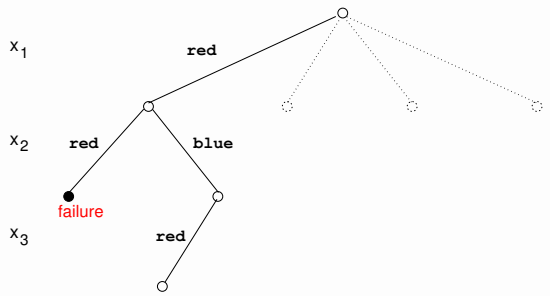
Example



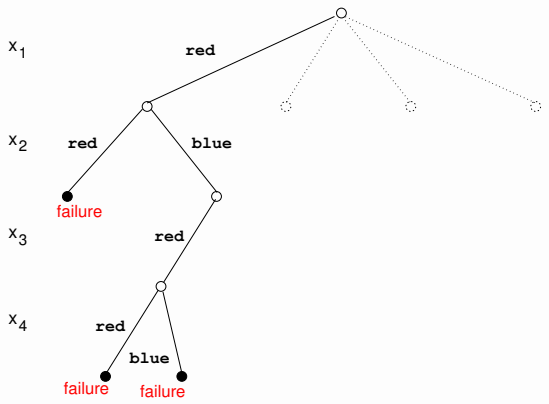
Example



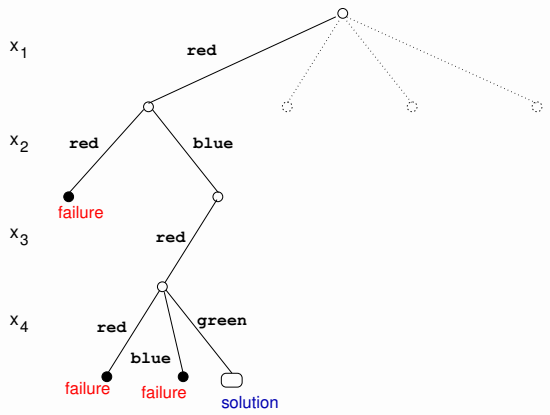
Example



Example



Example



Dynamic BT et similia

- Identifies the reasons of conflicts
- Remembers the conflicts by means of *no-goods*
- Not strictly chronological

References: See mainly works by Ginsberg. (Further tree-search techniques will be discussed later.)

Chronological BT

Observations:

- Variable and value ordering are crucial
- Static vs. dynamic ordering heuristics
- Disadvanages:
  - Thrashing (due to wrong initial choices)
  - Redundant work (reasons for failure are not remembered)
  - Detects conflicts too late

Consistency techniques

- Aimed at reducing the search space
- Constraints used *a priori*
- Remove combinations of assignments which cannot lead to a consistent solution
- Reasoning on constraint graph

▶ Based on the concept of consistency properties

Example

Let $A < B$ be a constraint between A with $D_A = \{3, \ldots, 7\}$ and B with $D_B = \{1, \ldots, 5\}$.

- for some values in D_A there does not exist a consistent value in D_B
- such values can be removed from the respective domains
- this reduction does not necessarily remove ALL inconsistent pairs (e.g., $A = 4$ and $B = 4$)

Node consistency

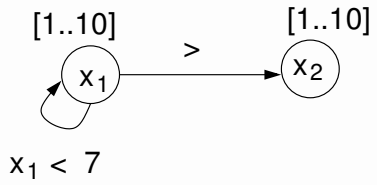
A constraint graph is node consistent if, for each node, each value is consistent with unary constraints.

Consistency classes

- **Node** consistency
- **Arc** consistency
- **Path** consistency
- **k-consistency**
- . . .

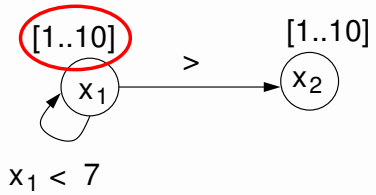
Node consistency

A constraint graph is node consistent if, for each node, each value is consistent with unary constraints.



Node consistency

A constraint graph is node consistent if, for each node, each value is consistent with unary constraints.



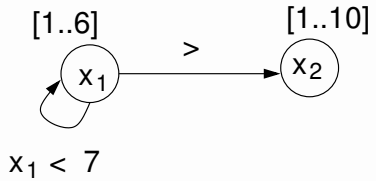
Node consistency

- Simplest form of consistency
- Only propagates information from unary constraints
- Node consistency filtering algorithm running in polynomial time

NC is not sufficient: There could be still domain values not consistent with the other constraints

Node consistency

A constraint graph is node consistent if, for each node, each value is consistent with unary constraints.

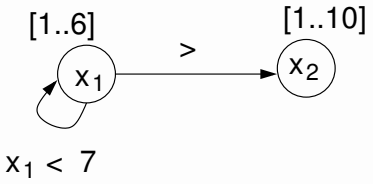


Arc consistency

A constraint graph is arc consistent if it is node consistent and, for each arc connecting variables x_i and x_j, for each value in the domain of x_i there exists a value in the domain of x_j consistent with binary constraints.

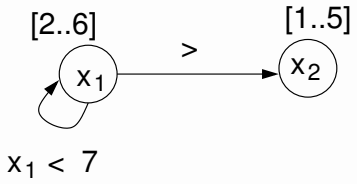
Arc consistency

A constraint graph is arc consistent if it is node consistent and, for each arc connecting variables x_i and x_j, for each value in the domain of x_i there exists a value in the domain of x_j consistent with binary constraints.



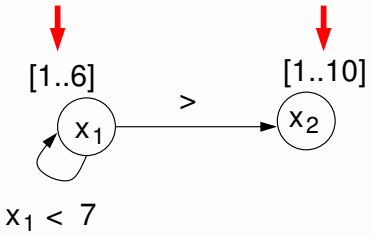
Arc consistency

A constraint graph is arc consistent if it is node consistent and, for each arc connecting variables x_i and x_j, for each value in the domain of x_i there exists a value in the domain of x_j consistent with binary constraints.



Arc consistency

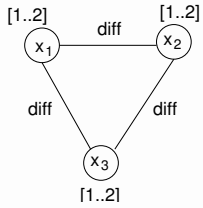
A constraint graph is arc consistent if it is node consistent and, for each arc connecting variables x_i and x_j, for each value in the domain of x_i there exists a value in the domain of x_j consistent with binary constraints.



Arc consistency

- Arc consistency filtering algorithms running in polynomial time
- Several, specialized algorithms available

AC is not sufficient.



Arc consistency

A CSP after achieving arc consistency:

- Domain size for each variable becomes one $\rightarrow$ exactly one solution exists
- Any domain becomes empty $\rightarrow$ no solution exists
- Otherwise $\rightarrow$???

In summary

We can solve a CSP by combining propagation and search:

- Applying filtering algorithms until no further domain value deletion is possible
- If all domains of size one $\rightarrow$ one solution
- Else if any domain empty $\rightarrow$ no solution
- Else search in the reduced domains

K-consistency

- Generalizes the concept of arc consistency (equivalent to 2-consistency)
- Complete N-consistency can be achieved in exponential time
- Not used in practice, except for some specific cases
- Better to resort to *incomplete* propagation (i.e., not all the inconsistent values are removed)

In summary

We can solve a CSP by combining propagation and search:

- Applying filtering algorithms until no further domain value deletion is possible
- If all domains of size one $\rightarrow$ one solution
- Else if any domain empty $\rightarrow$ no solution
- Else search in the reduced domains

We are now ready for CP!

Finite domains CP

- One to one mapping with CSP concepts
- Problem modelling
  - Variables range on a finite domain of objects of arbitrary type
  - Constraints among variables
    - mathematical constraints
    - symbolic constraints

Constraints

- Mathematical constraints: $=, >, <, \neq, \leq, \geq$
  - Propagation: arc-consistency
- Symbolic Constraints (aka Global constraints)
  - Capture a sub-problem with well identifiable structure
  - Embed more powerful propagation algorithms
  - More concise formulation
  - Also user defined

Finite domains CP

- Problem solving
  - Propagation algorithms embedded in constraints
  - Arc consistency as standard propagation
  - More sophisticated propagation for *global constraints*
  - Search strategies

Symbolic constraints

Alldifferent: $alldiff([x_1, \ldots, x_n])$

- Holds iff all the variables involved are assigned to different values
- Equivalent to $n(n-1)/2$ binary constraints $x_i \neq x_j$
- Complete filtering algorithms in polynomial time

Example: Sudoku can be described by posting $9+9+9$ $alldiff$ constraints.

Symbolic constraints (cnt.)

▶ Deeper propagation than arc-consistency, even if the propagation is not complete.

Example:

- $x_1, \ldots, x_4$, with $D_1, \ldots, D_3 = \{1,2,3\}$ and $D_4 = \{1,2,3,4\}$
- $alldiff([x_1, x_2, x_3, x_4])$

▷ Arc-consistency: no propagation

▷ Alldifferent: values $1,2,3$ are deleted from D_4

Symbolic constraints (cnt

Global cardinality:
$gcc([x_1, \ldots, x_n], [v_1, \ldots, v_m], [l_1, \ldots, l_m], [u_1, \ldots, u_m])$

- Holds iff the number of variables in $[x_1, \ldots, x_n]$ which assume value v_i is within l_i and u_i
- Complete filtering algorithms in polynomial time and very efficient bound propagation

Example: rostering, car-sequencing.

Example

N-Queens

- Variables: $x_i \in \{1, 2, \ldots, N\}$ is the row of queen in column i
- Constraints:
  - $alldiff([x_1, \ldots, x_n])$
  - $alldiff([x_1 - 1, \ldots, x_n - n])$
  - $alldiff([x_1 + 1, \ldots, x_n + n])$

Symbolic constraints (cnt

- Cycle
- Stretch
- Pattern
- Element
- Edge finder
- ...

▶ Not always filtering algorithms are complete.

Constraints: remarks

- Tradeoff between propagation and computational effort
- Generalization of frequently found constraints
- Concise and easily understandable code
- Symbolic constraints represent independent subproblems (relaxations of the original problem)
- Symbolic Constraints are available in most CP tools

Solving

- Iterative process between propagation and search (i.e., assign temptative values)
- Interaction among constraints
- Search strategy is independent

Constraint: Modelling

Two further important constraint classes:

- **Implied (or redundant)** constr.: not needed to model the problem, but make propagation more efficient
- **Symmetry-Breaking** constr.: reduce the search space by cutting *symmetric* states
  E.g.: Given a solution to N-Queens problem, we can obtain other equivalent solutions by rotation of the chessboard

Solving

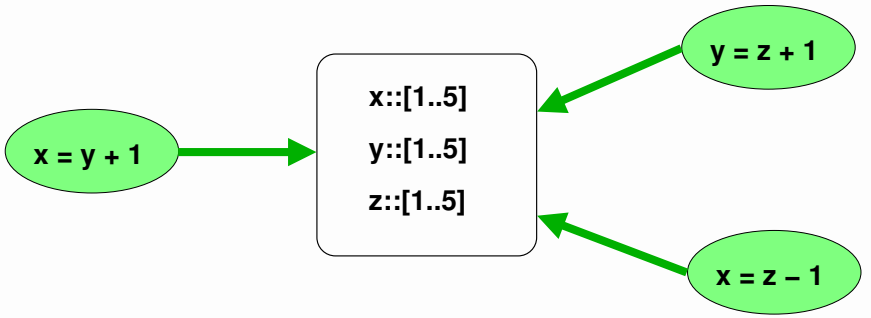
In practice:

- Constraints propagate as much as possible and reduce variable domains
- If no termination (conditions as above) performs one ore more search steps by assigning values
- Constraints are triggered by the assignments and propagate again
- . . .and so forth. . .

Constraint interaction

- Interact each other through shared variables in the *constraint store*
- Trigger propagation each time an event is raised on a variable x
  - a change in the domain of x
  - a change in the range of the domain of x
  - assignment of variable x to a value

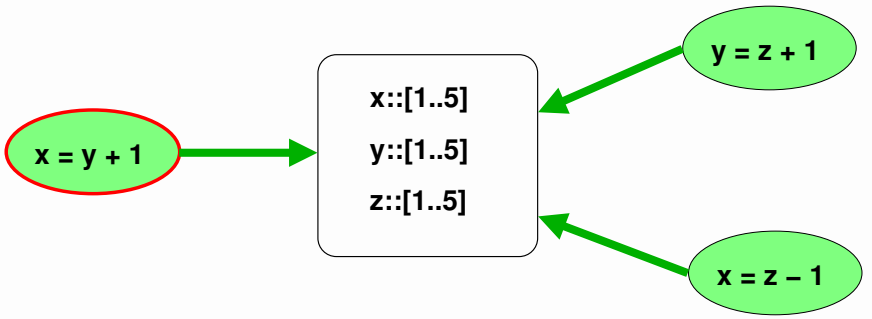
Example



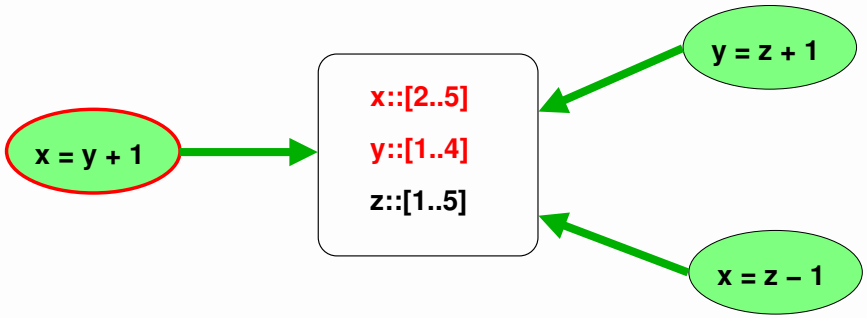
Constraint interaction

- In general each variable is involved in many constraints
- Each change in variable domains as a result of propagation may result in further propagations to other variables
- **Constraint agents**: during their lifetime they alternate between suspended and waking states (triggered by events)

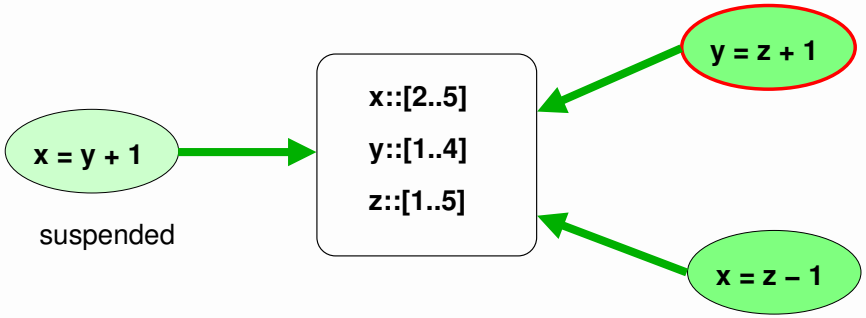
Example



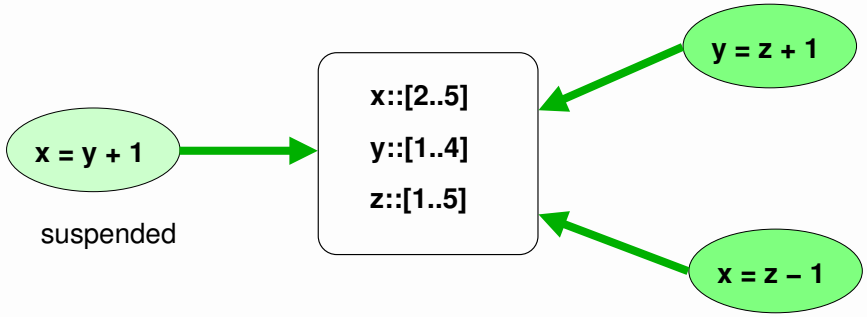
Example



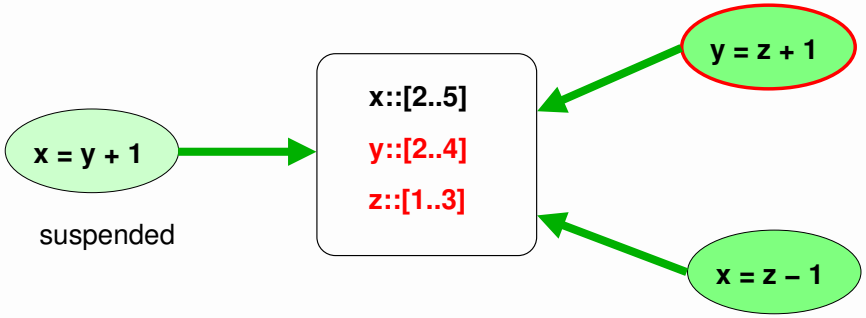
Example



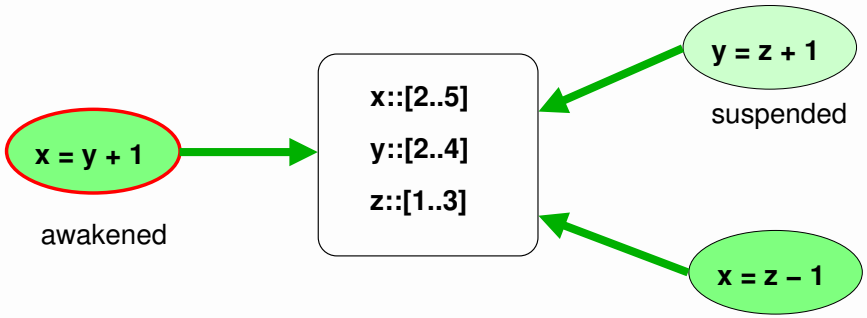
Example



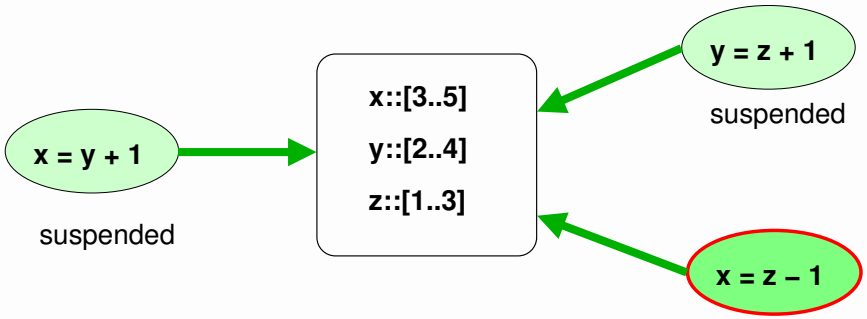
Example



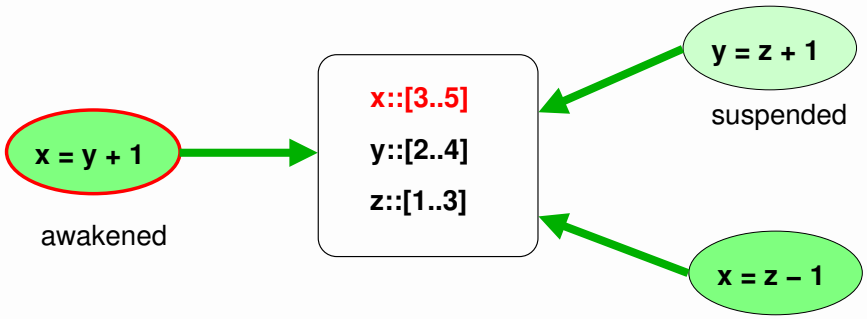
Example



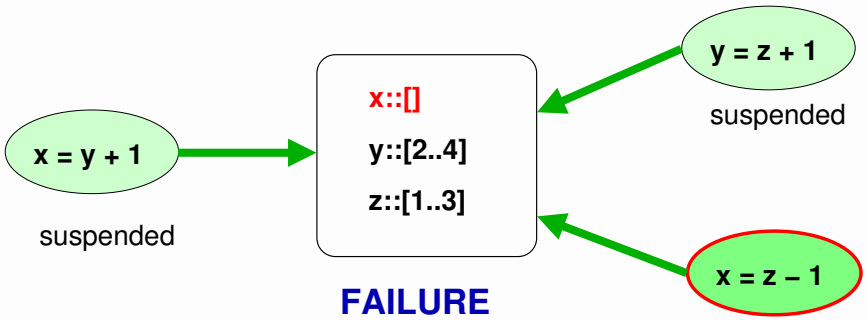
Example



Example



Example



Constraint interaction

Important:

- The order in which constraints are considered (delayed and awakened) does not affect the propagation results
- **but** can affect the performance of the algorithm

Search

Basic and general idea:

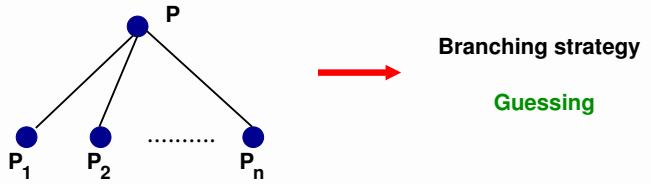
- Divide the problem into subproblems and solve them independently
- Subproblems must partition the original problem

Search

Propagation is, in general, not complete. After propagation:

- Solution found $\rightarrow$ stop
- Failure detected $\rightarrow$ backtracking
- Domains contain some values $\rightarrow$ SEARCH

Search



- Branching strategies define the way of partitioning the problem P into easier subproblems $P_1, P_2, \ldots, P_n$
- To each subproblem: apply again propagation. New branches can be pruned by means of information derived from the branching

Search

Most popular branching in CP: **labelling**

- Select one VARIABLE
- Select one VALUE in its domain
- Assign the VALUE to the VARIABLE

- The order variables and values are chosen greatly affects the performances of the search algorithm

Variable choice

Principle: ***More difficult variables first***

- **First Fail**: select first the variable with the smallest domain
- **Most Constraining Principle**: select first the variable involved in the greatest number of constraints
- Problem dependent heuristics

Var and value choice

- Static
- Dynamic
- General search heuristics (e.g., *first fail* principle)
- Problem-dependent heuristics
- Learning (e.g., in ACO)

Value choice

Principle: ***More promising values first***

- **Least Constraining Principle**: prefer values with maximal support (remaining values for uninstantiated vars)
- Problem dependent heuristics

Constraints again

- Constraints posted by the problem model or which cut part of the search tree not to be explored **remain always valid**
- The propagation of constraints performed upon variable instantiation **is retracted in case of backtrack**

LDS

- Widely used in CP
- Several variants

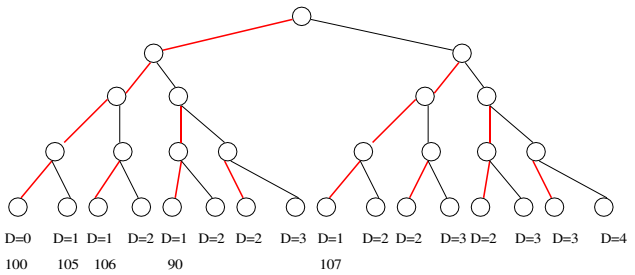
Hypothesis: A good heuristic makes few mistakes

Advanced strategies

- Dynamic heuristics
- Non chronological backtracking
- Learning (variable/value ordering, no-goods, etc.)
- Limited Discrepancy Search (LDS) and variants

LDS

LDS explores leaves at increasing discrepancy from the *reference* one, i.e., the one suggested by the heuristic.



LDS

Variants:

- Depth-Bounded Discrepancy Search
- Weighted Discrepancy Search
- Climbing Discrepancy Search

CP and Optimization

- CP alone is not efficient in optimizing (bound constraint is too loose)
- Many successful examples of integration of CP and OR optimization techniques
- Special global constraints taking advantage of the information on the cost function

CP and Optimization

- Combination with Branch & Bound
- A constraint on the objective value is posted

Each time a solution is found with cost c^*, impose a constraint on the remaining search tree, stating that further solutions (of cost c) must be better than the best one found so far: $c < c^*$.

CP and Metaheuristics

- Metaheuristics are applied before systematic methods, providing a valuable input, or vice versa.
- Metaheuristics use CP and/or tree search to efficiently explore the neighborhood.
- Metaheuristic concepts can be used to obtain incomplete but efficient tree exploration strategies (e.g., non-chronological backtracking in a TS fashion)

References

- Lecture notes and slides of the 1st summer school on CP: `www.math.unipd.it/~frossi/cp-school/`
- M.Milano (ed.). **Constraint and integer programming**. Kluwer, 2004.
- F.Focacci, F.Laburthe and A.Lodi. **Local search and constraint programming**, in Handbook of Metaheuristics, Kluwer, 2002.
- P.Van Hentenryck and L.Michel. **Constraint-based local search**. The MIT Press, 2005.